**Mocking and Stubbing**

**Mockito**

Vaishnavi

28/6/2025

This project demonstrates the use of mocking and stubbing in unit testing with Mockito, where a WeatherApi interface is mocked to simulate real weather data responses. The goal is to isolate the WeatherService logic and test its behavior without relying on an actual API.

**Objective:**

* **Mock External Dependencies:** Replace the real WeatherApi with a mock object to avoid external API calls during testing.
* **Stub Return Values:** Define specific return values for different city inputs using when(...).thenReturn(...) to simulate real-world data.
* **Validate Service Logic:** Ensure WeatherService formats and returns correct messages based on mocked API responses.

**Implementation:**

### Create a Maven Java Project

**IntelliJ IDEA**: File → New → Project → Maven

### Add Mockito and JUnit to Your Project

#### By using ****Maven****: add the dependencies in pom.xml

<!-- JUnit 5 -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.6.0</version>

<scope>test</scope>

</dependency>

**Create a Interface for API, Service that uses API and Test Class using Mockito**

**WeatherApi.java:**

public interface WeatherApi {

String getWeather(String city);

}

**WeatherService.java:**

public class WeatherService {

private WeatherApi api;

public WeatherService(WeatherApi api) {

this.api = api;

}

public String fetchTodayWeather(String city) {

return "Today's weather in " + city + ": " + api.getWeather(city);

}

}

**WeatherServiceTest.java:**

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

public class WeatherServiceTest {

@Test

public void testWeatherMultipleCities() {

WeatherApi mockApi = mock(WeatherApi.class);

when(mockApi.getWeather("Chennai")).thenReturn("Sunny, 32°C");

when(mockApi.getWeather("Delhi")).thenReturn("Cloudy, 28°C");

when(mockApi.getWeather("Mumbai")).thenReturn("Rainy, 29°C");

when(mockApi.getWeather("Kolkata")).thenReturn("Humid, 31°C");

when(mockApi.getWeather("Bengaluru")).thenReturn("Cool, 24°C");

WeatherService service = new WeatherService(mockApi);

assertEquals("Today's weather in Chennai: Sunny, 32°C", service.fetchTodayWeather("Chennai"));

assertEquals("Today's weather in Delhi: Cloudy, 28°C", service.fetchTodayWeather("Delhi"));

assertEquals("Today's weather in Mumbai: Rainy, 29°C", service.fetchTodayWeather("Mumbai"));

assertEquals("Today's weather in Kolkata: Humid, 31°C", service.fetchTodayWeather("Kolkata"));

assertEquals("Today's weather in Bengaluru: Cool, 24°C", service.fetchTodayWeather("Bengaluru"));

System.out.println(service.fetchTodayWeather("Chennai"));

System.out.println(service.fetchTodayWeather("Delhi"));

System.out.println(service.fetchTodayWeather("Mumbai"));

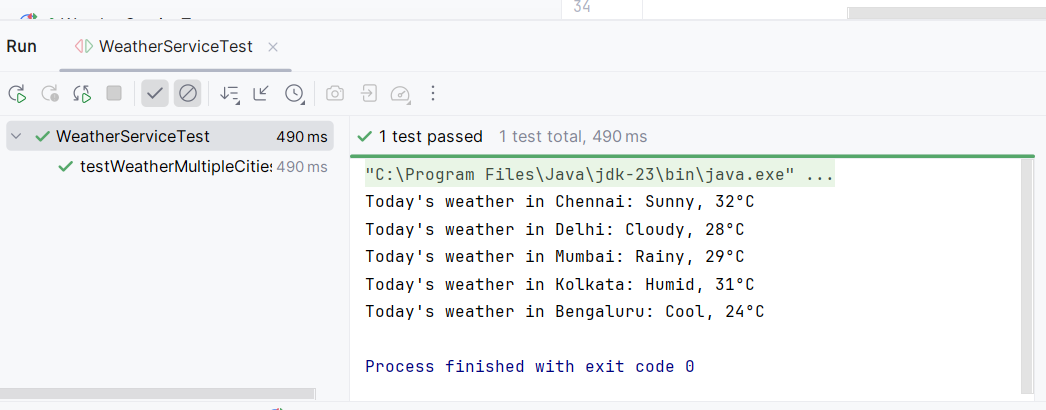
System.out.println(service.fetchTodayWeather("Kolkata"));

System.out.println(service.fetchTodayWeather("Bengaluru"));

}

}

**Output:**

****